Power Systems

IBM

IBM GZIP Hardware Accelerator

Frank Haverkamp haverkam@de.ibm.com
Joerg-Stephan Vogt jsvogt@de.ibm.com
Wolfgang Bolz wolfgang.bolz@de.ibm.com
Burkhard Steinmacher-Burow steinmac@de.ibm.com

July 2016 – Version 0.2

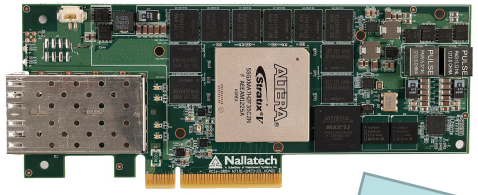
GZIP Hardware Acceleration

- Introduction
- Examples
- How to use it?
- CAPI Example

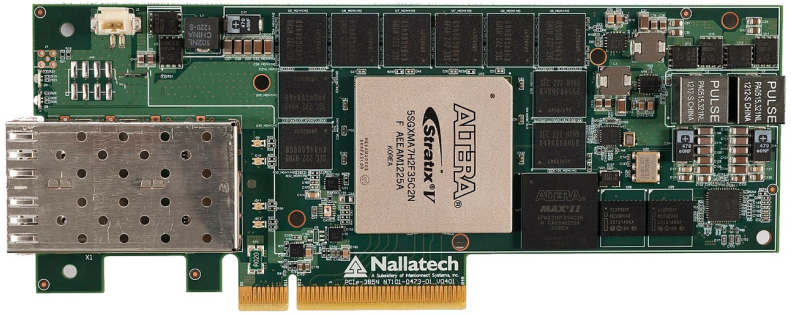
- Summary

Evolution

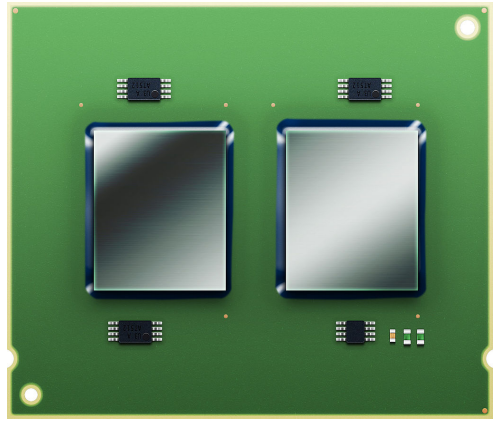
GenWQE/PCIe GZIP Accelerator
FC #EJ12, #EJ13



CAPI GZIP Compression Solution Architecture
#EJ1A & EJ1B

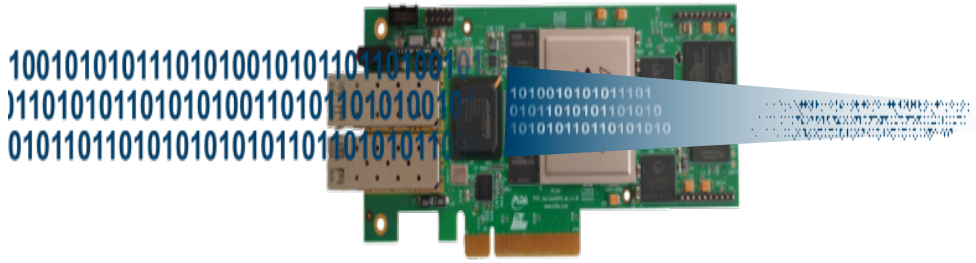


On CPU Accelerator?



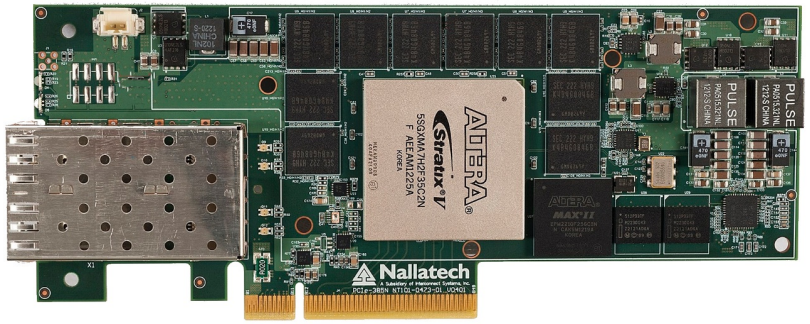
GZIP - Overview

- **Business value**
  - High throughput compression
    Saves storage and I/O bandwidth with little or no overhead
  - CPU offload, CAPI interface with negligible software load
    Frees up CPU cores for higher value computation or licensed software
  - Lower power consumption by offloading the CPU intensive compression to an FPGA
- **Features**
  - DEFLATE standard format, widely used for data interchange (RFC1950 zlib, RFC1951 deflate and RFC1952 gzip)
  - ~ 1.8 GiB/sec compression throughput
  - ~ 2GiB/sec decompressed throughput
  - 4 – 30 x speed-up achievable
  - Compression ratio near software zlib/gzip
- **Example use cases**
  - Genomics
  - Datacenter/Cloud
  - Backup solutions

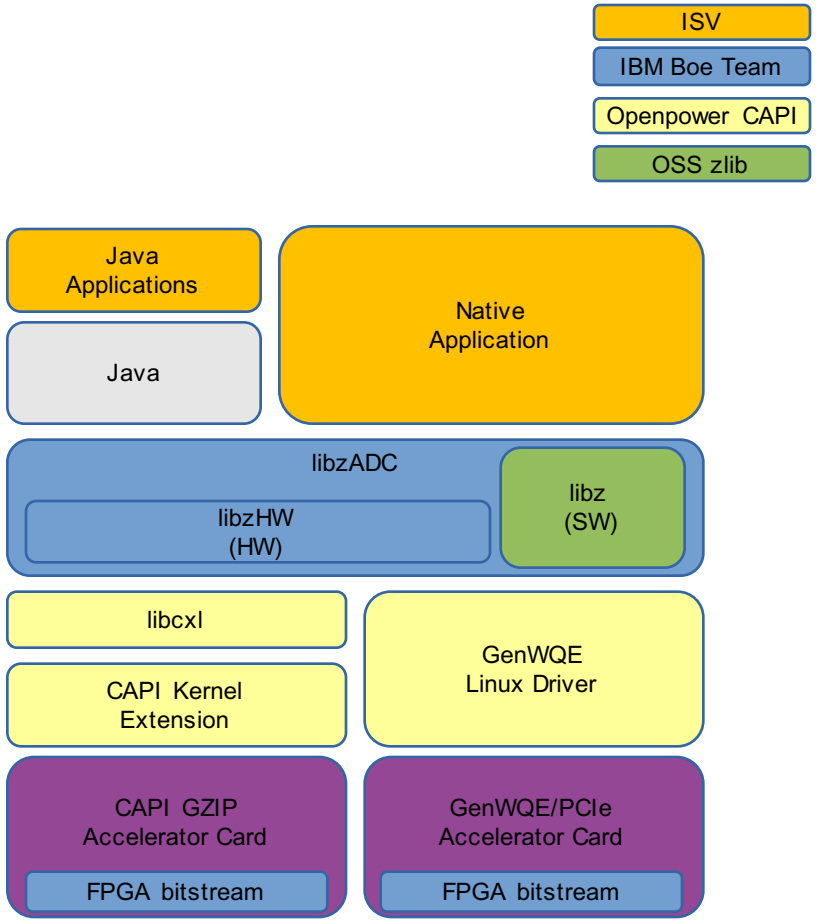


GZIP Compression Solution Architecture

- GenWQE/PCIe
  - FC #EJ12, #EJ13
- CAPI GZIP
  - Power8 and OpenPower enabled: Tuleta-L, Habanero-LC, Firestone-LC (bare-metal only)
  - Ubuntu 14.04.5, RHEL 7.2 LE
  - FC #EJ1A & EJ1B



Hardware platform
Altera Stratix V FPGA



Software stack (Linux on Power)

GZIP: DEFLATE Algorithm

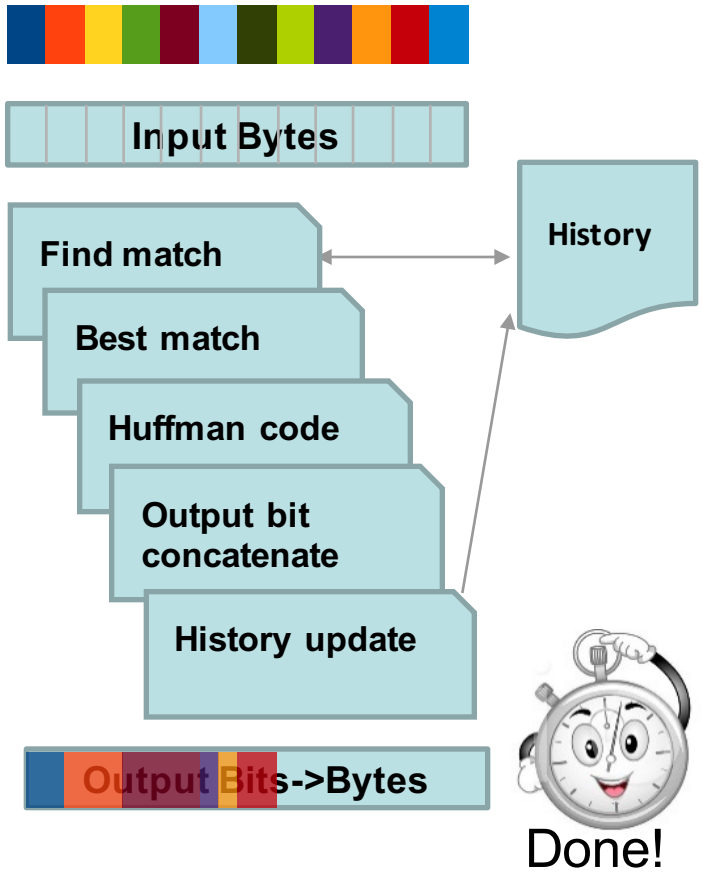
Compression with Lempel-Ziv (LZ) methods aims at finding repeated byte sequences in the data. It then replaces the repeated byte sequence by a backwards reference (sequence length, distance) to the original occurrence. Unmatched bytes are coded as literals. A subsequent Huffman coder then usually codes the symbols (literals and references) with bit sequences which are short for frequent codes, and longer for less frequent ones.

The CAPI GZIP compression implementation consists of three main stages:

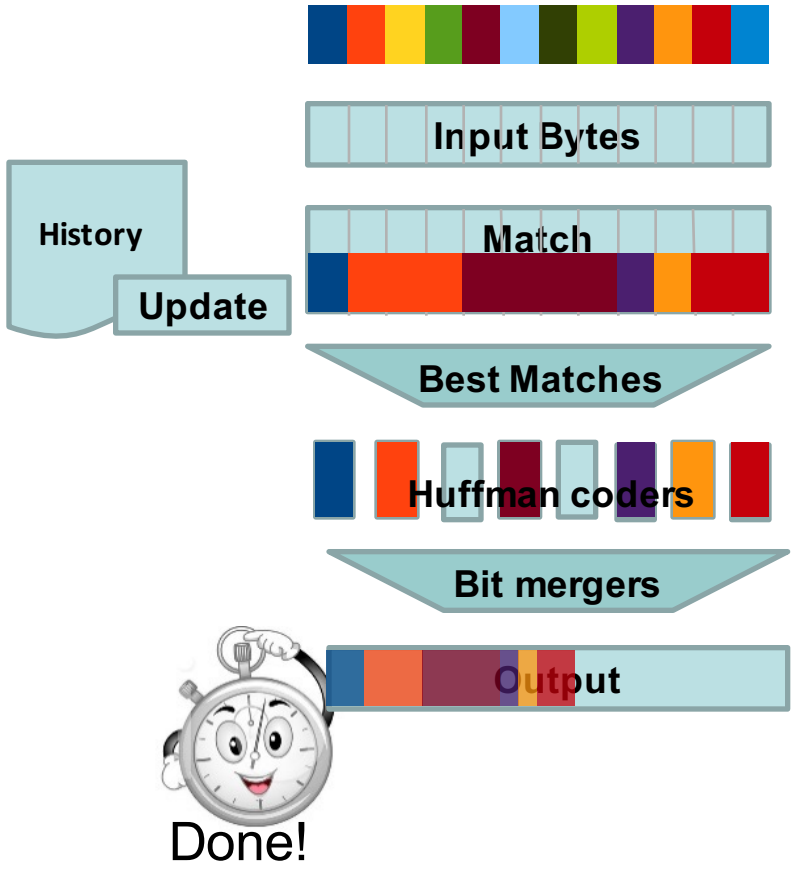
1. The first stage uses two sets of hashes to find previous occurrences of byte sequences in the input data stream. The best match, i.e. longest byte sequence with shortest distance, gets selected and overlapping matches are removed. Due to the nature of hashes, there may be hash collisions. The current implementation, for example, can find at most the previous two of such colliding sequences. Every clock cycle this stage outputs a sequence of 2-12 literals (single bytes) or backwards references (3-11 byte length matches with a distance to the previous occurrence).
2. The second stage picks one of the matches per cycle and compares it with the original data to search for longer matches. If this extender stage finds a longer match, that match will replace the original match from the first stage.
3. The third stage encodes the literals and matches into Huffman codes. The encoder has multiple sets of Huffman codes to choose from. Based on the initial few kilobytes of data it selects which of the sets is most suitable.

Competition

Software



FPGA



Multi-threading/Multi-processing/Hardware Virtualization

Feature	GenWQE/PCIe	CAPI GZIP
Multi-processing	Yes	Up to 512 processes
Multi-threading	Yes	Yes
PCI pass-through	Yes	Not yet supported
SRIOV	Yes (for System z) Untested (for System p)	No
Docker	Yes	Untested
Multiple cards in one system	Yes • Automatic assignment • Fixed assignment	Yes • Automatic assignment • Fixed assignment

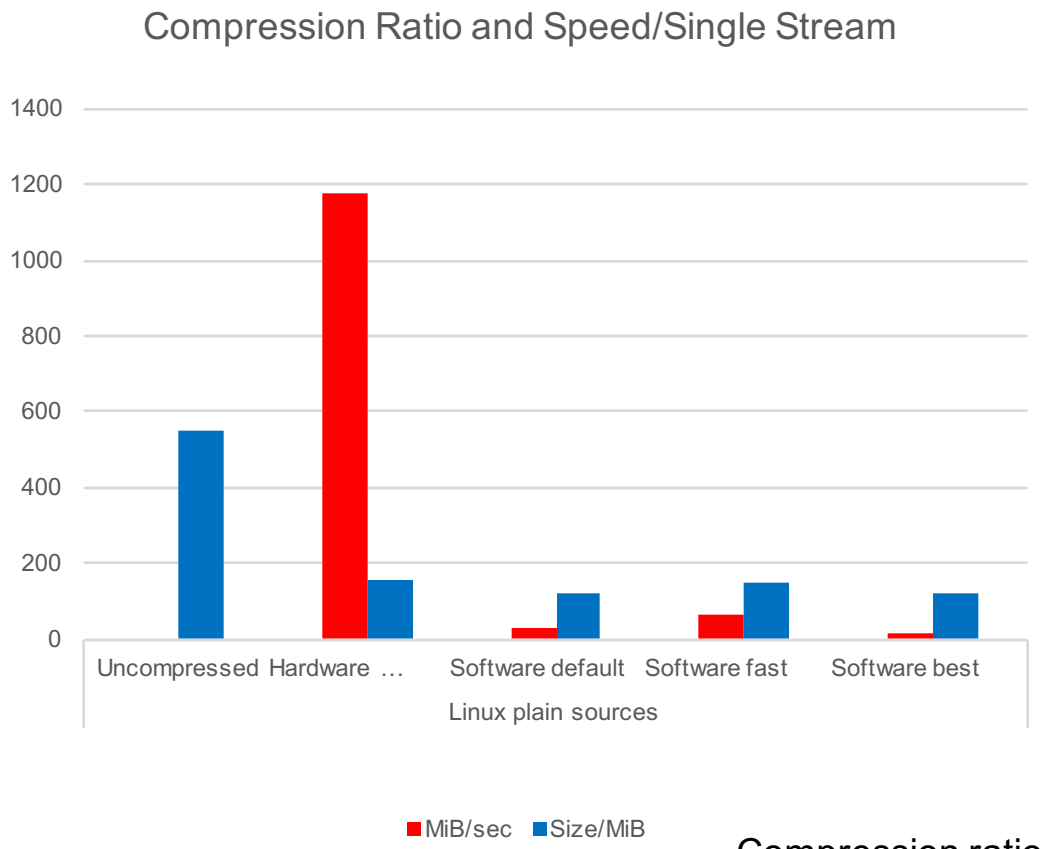
Reliability Availability Serviceability (RAS)

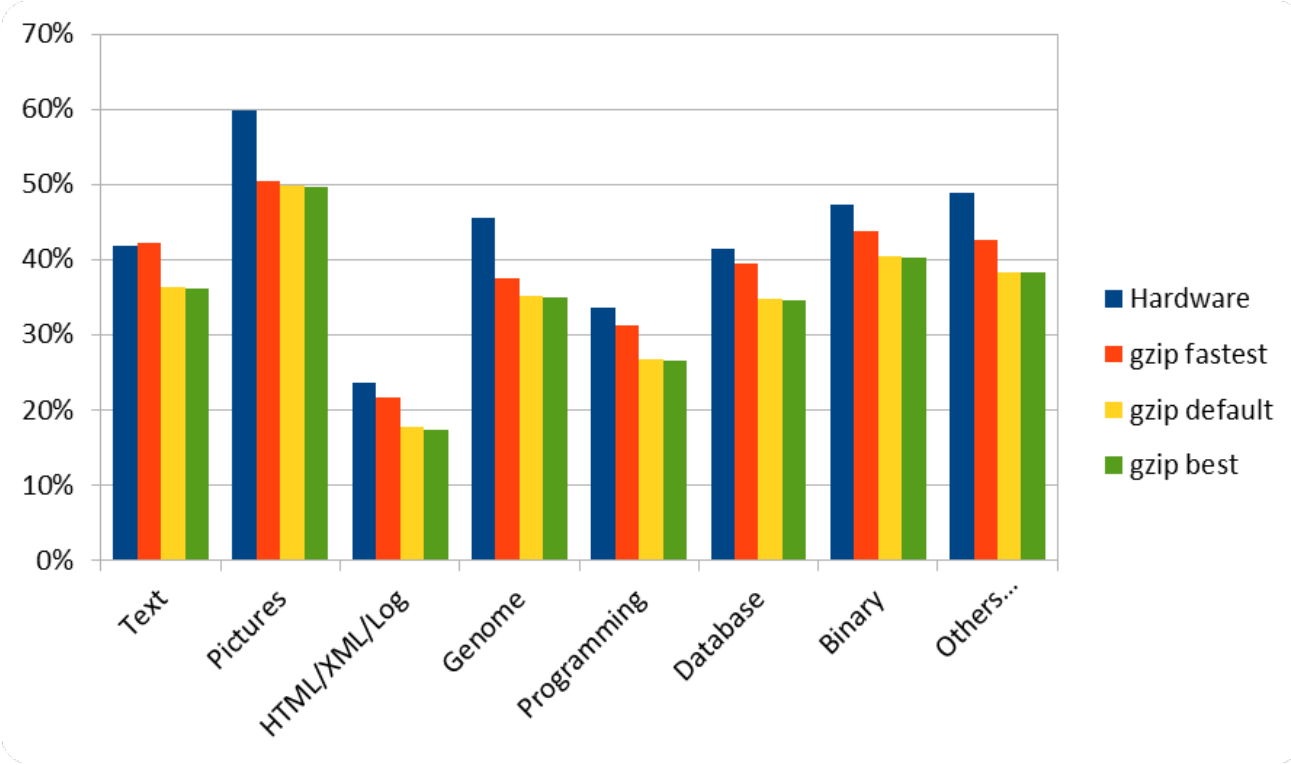
GenWQE/PCIe		
Hardware	Compression	Decompression
	• Compressed data is decompressed and compared to original data • Parity checking on busses	• Redundant implementation with checking • Parity checking on busses
Software	• Retry on alternate card (if available) • Abort with appropriate error code	• Retry on alternate card (if available) • Abort with appropriate error code

CAPI GZIP		
Hardware	Compression	Decompression
	• Compressed data is decompressed and compared to original data • Parity checking on busses	• Redundant implementation with checking • Parity checking on busses
Software	• Abort with appropriate error code	• Abort with appropriate error code

Compression Ratio

Hardware and Software compression ratios are comparable however hardware compression delivers up to 20 x throughput (best with multiple threads). No need to modify applications but adjusting buffer sizes can improve performance significantly.

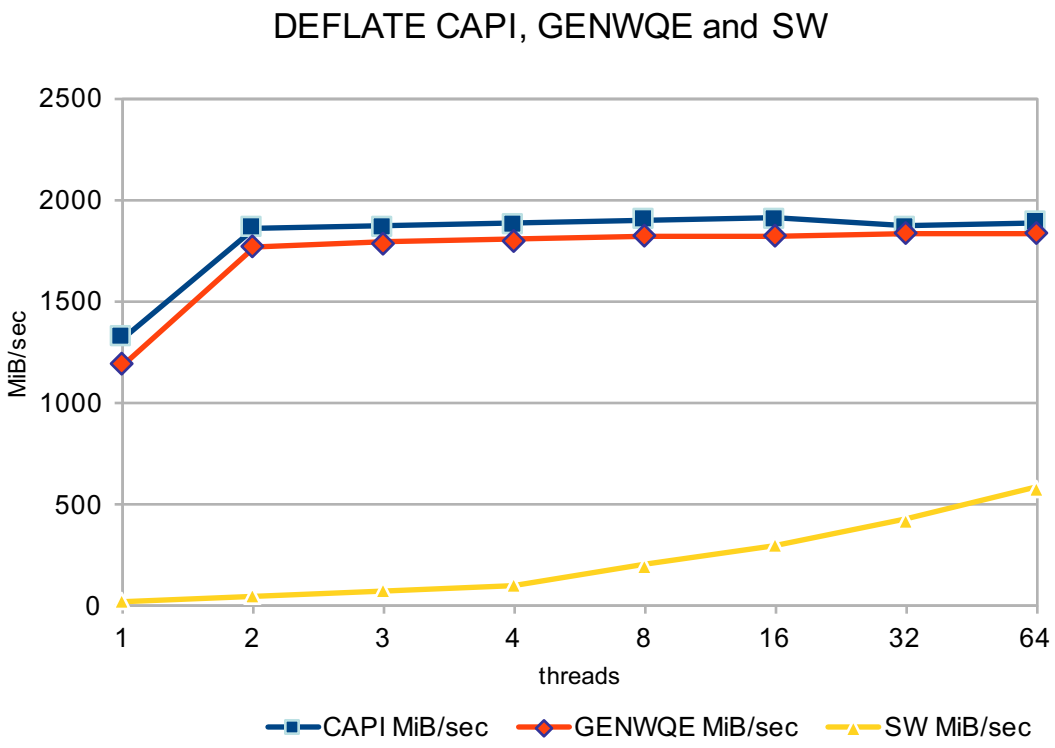


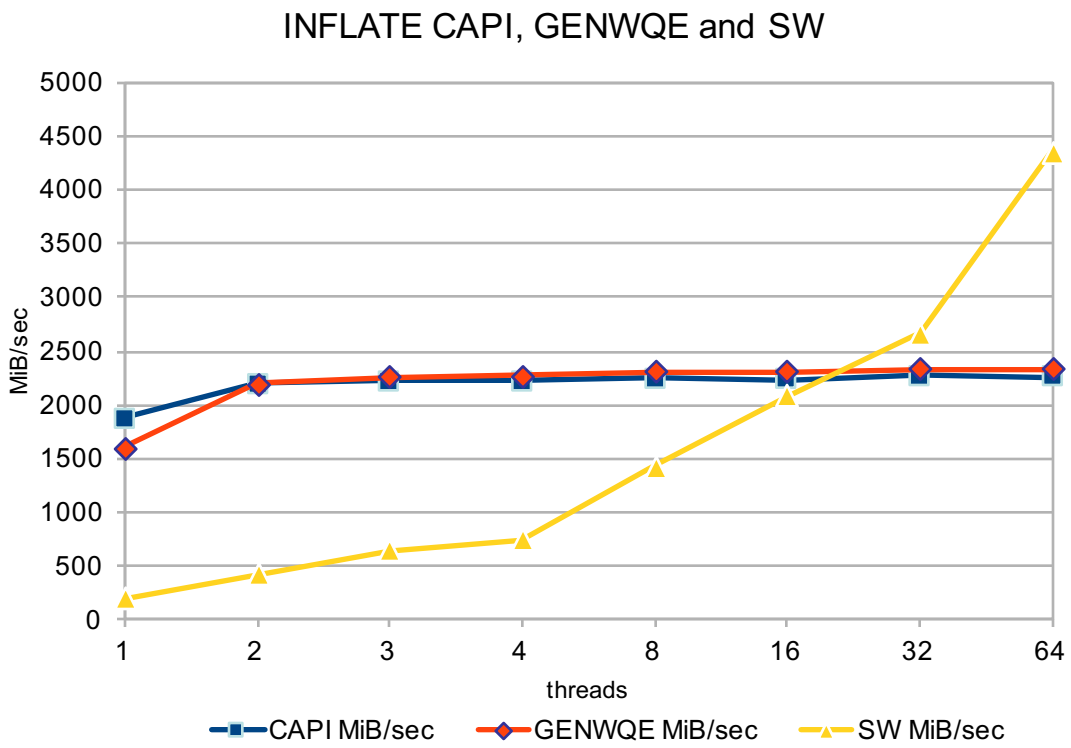


Compression ratio and speed – run 2 parallel threads to saturate hardware @~2GB/sec
https://github.com/ibm-genwqe/genwqe-user/blob/master/misc/ratio_test.sh

Example 1 – Multithreading/Throughput

Compressing and decompressing in parallel by multiple CPU threads/streams





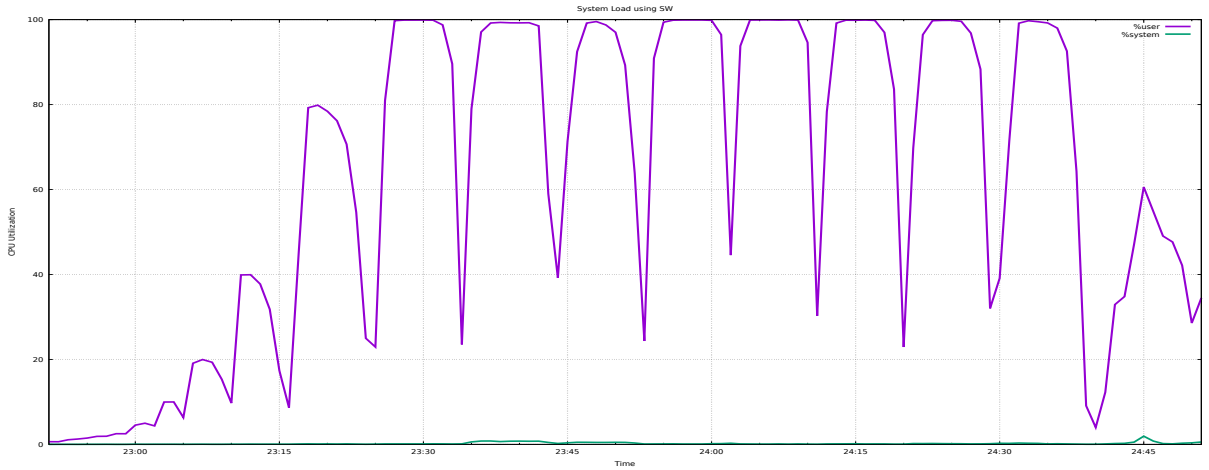
Maximum hardware throughput reached already with 2 threads
32 software deflate threads cannot beat CAPI accelerator
https://github.com/ibm-genwqe/genwqe-user/blob/master/tools/genwqe_mt_perf

Testsetup: IBM Tuleta 8247-22L
20 CPU cores @ 8 threads each
Avg: 3.694 GHz
Bitstream: 0x06030703475a4950

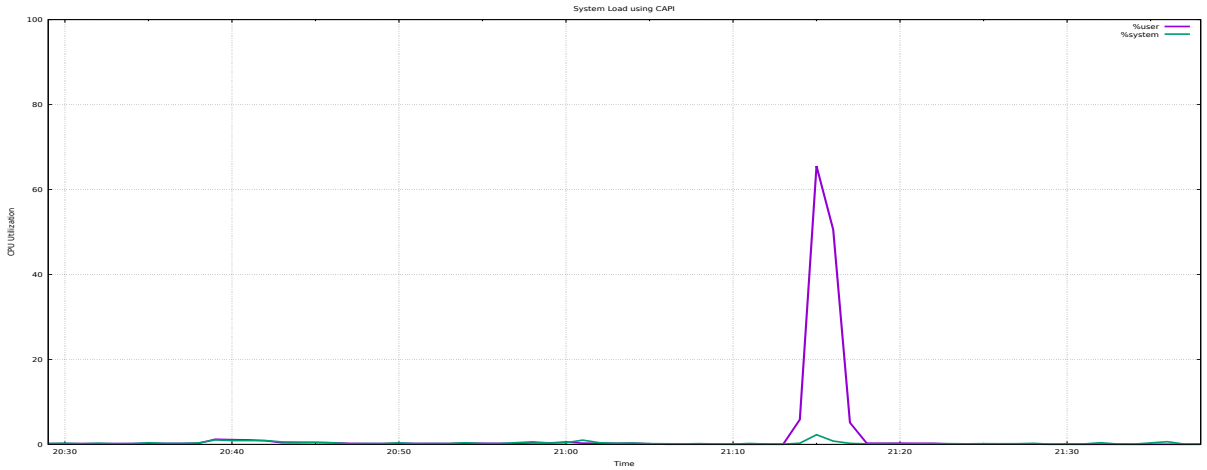
CPU Load

Compressing and decompressing in parallel by multiple CPU threads/streams

Software



CAPI GZIP Acceleration

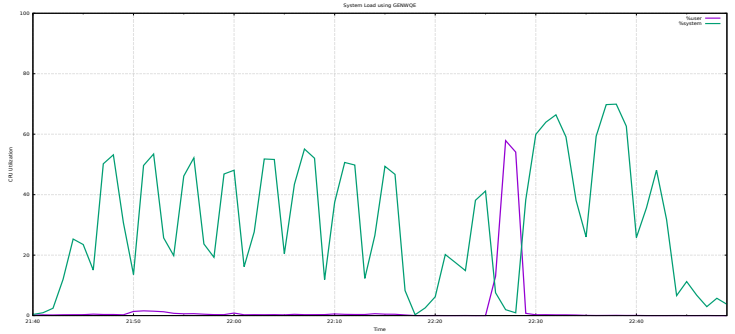


*) Software fallback due to small buffers

Drastic speed increase at almost no CPU load with CAPI accelerator!

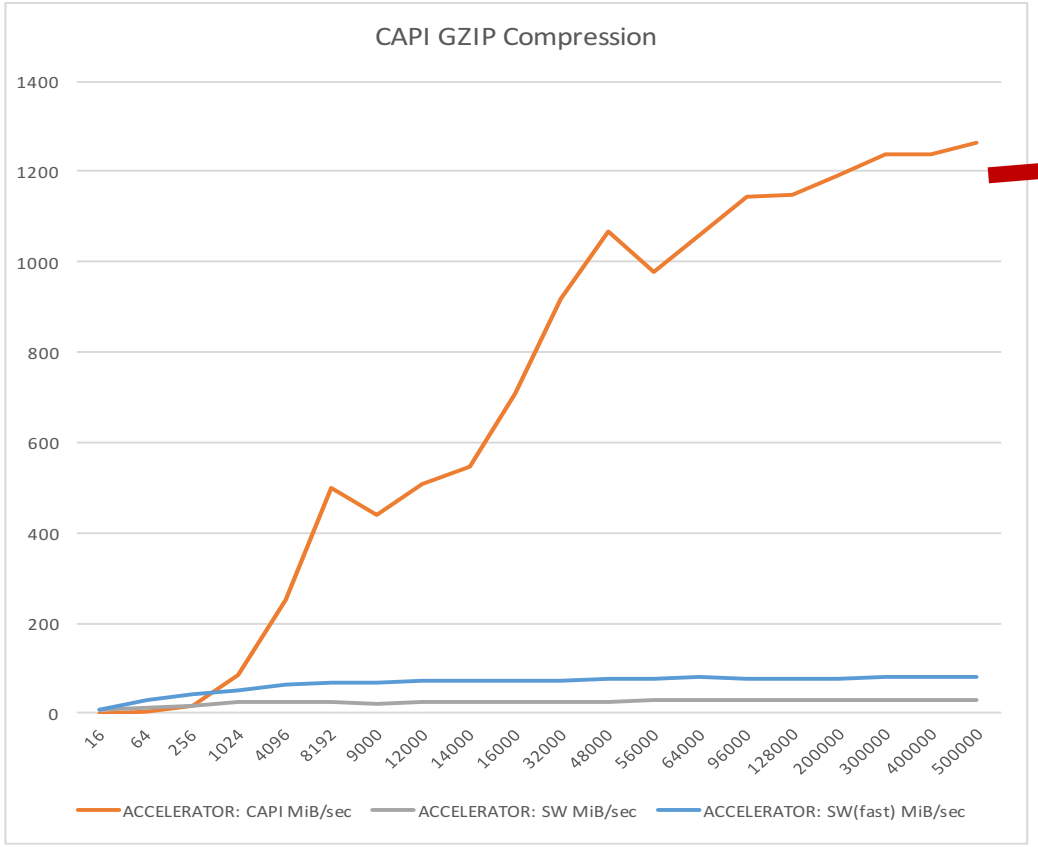
Data: http://corpus.canterbury.ac.nz/resources/cantrbry.tar.gz

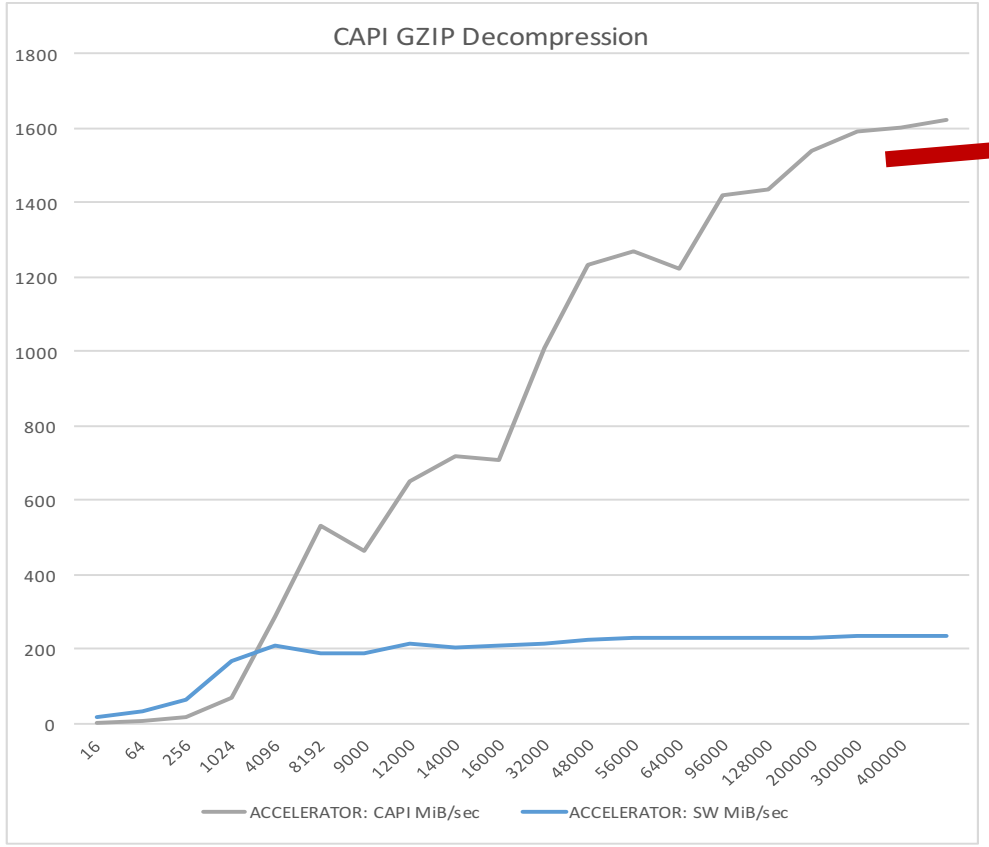
GenWQE/PCIe GZIP Acceleration



Example 2 – Files: gzip/gunzip replacement

Compression / decompression performance compared for different filesizes (CAPI vs SW default/fast)

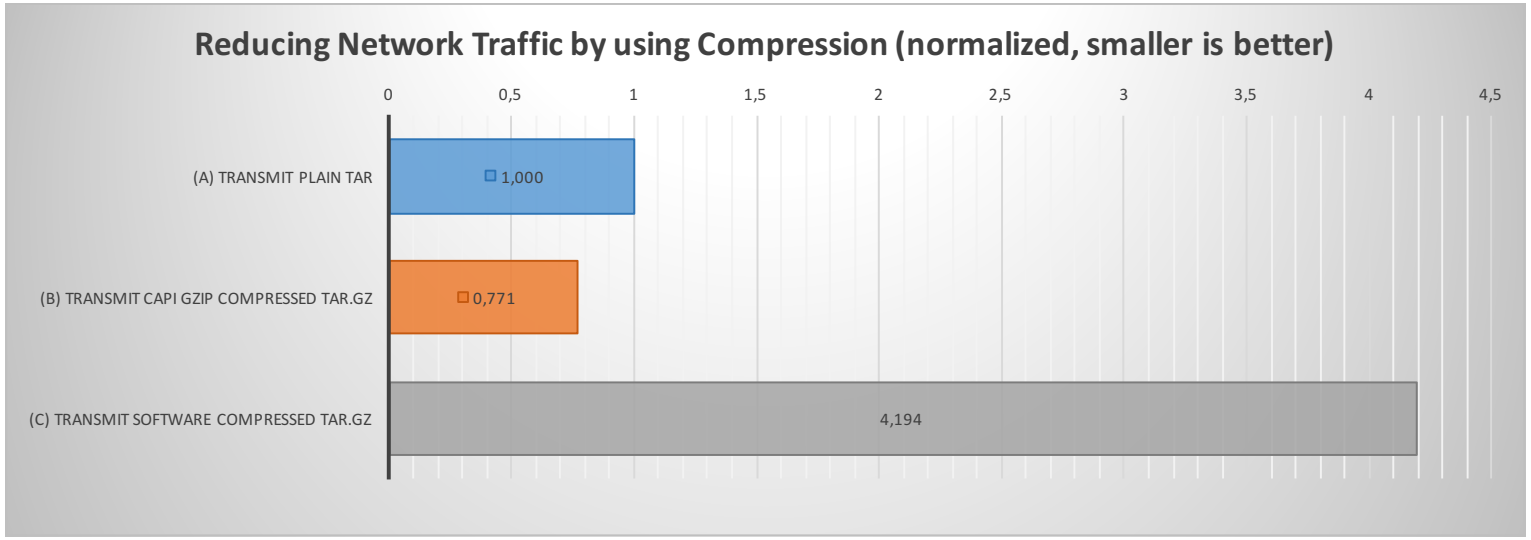




Keeping throughput up when pushing even more data – see customer feedback!
https://github.com/ibm-genwqe/genwqe-user/blob/master/tools/genwqe_file_perf

Example 3 – Network

Reduce network bandwidth and storage space by using CAPI GZIP acceleration e.g. in the Cloud



2 IBM Tuletas 8247-22L via 1Gb/sec Ethernet, 2.3 GiB Linux kernel git (text + binary)

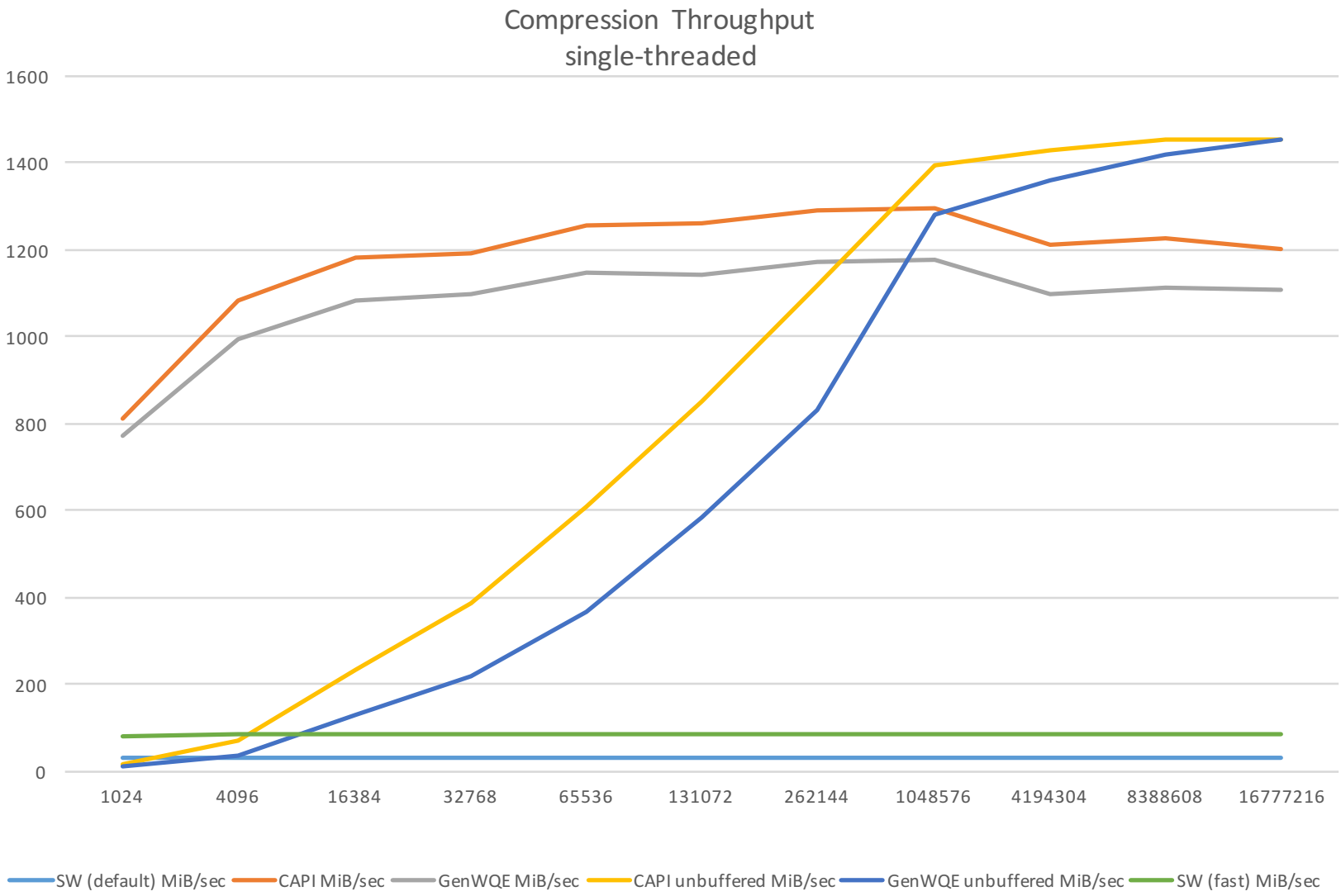
(A) Transmit uncompressed data

(B) Compress with CAPI and store compressed data => **Fastest & Saving disk-space at destination!**

(C) Compress with SW and store compressed data

https://github.com/ibm-genwqe/genwqe-user/blob/master/misc/netcat_test.sh

Compression: Influence of buffer size

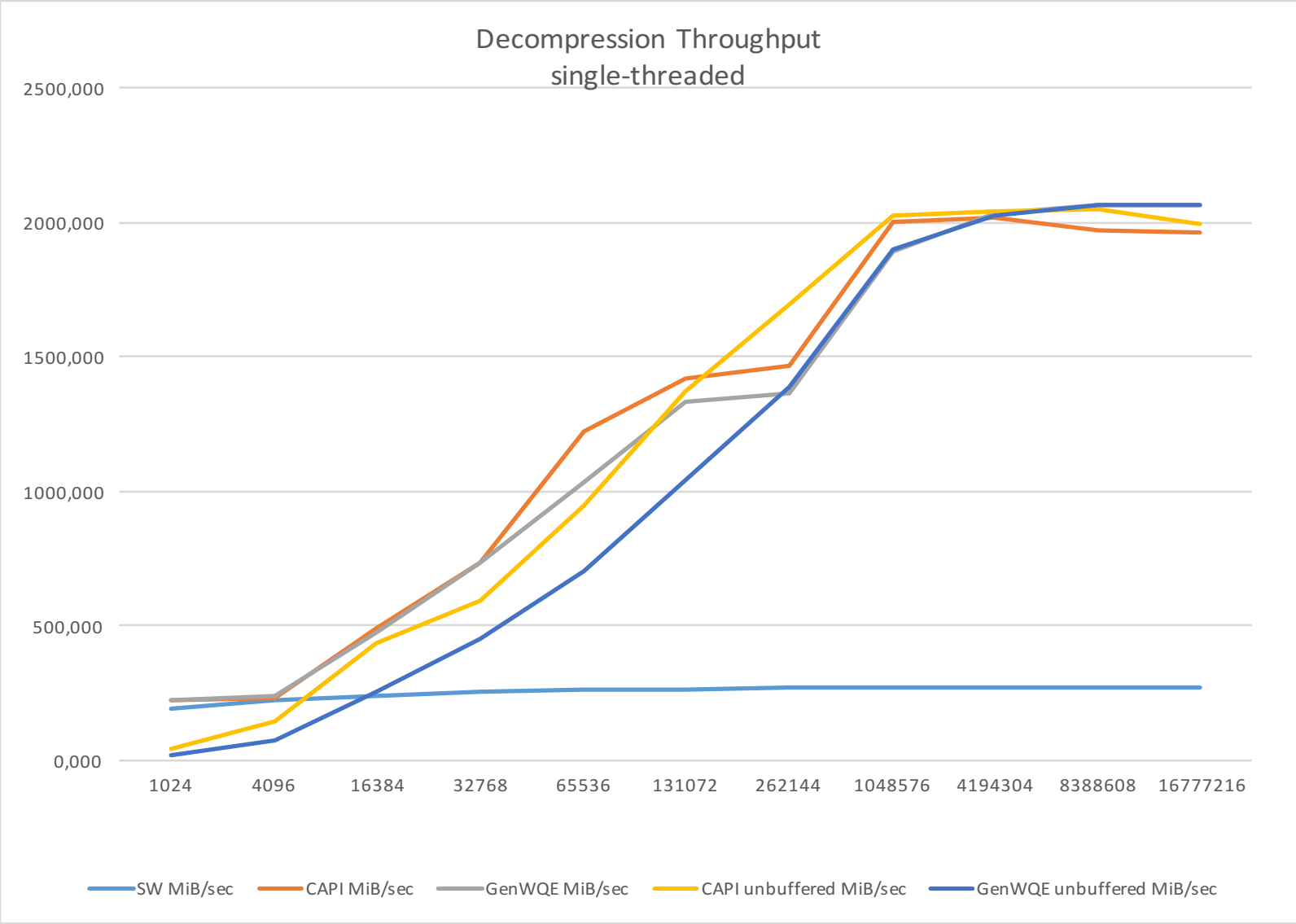


Compress linux.tar file, which is split into multiple buffers of different sizes (x-axis).

Buffering for deflate works well.

Compression is done always with hardware.

Decompression: Influence of buffer size



Decompress linux.tar.gz file, which is split into multiple buffers of different sizes (x-axis).

Buffering for inflate is limited.

Decompression is done with hardware if first input buffer is below threshold (currently16KiB).

SAM/Bam Formats

Field	Description	Type	Value
List of compression blocks (until the end of the file)			
ID1	gzip IDentifier1	uint8_t	31
ID2	gzip IDentifier2	uint8_t	139
CM	gzip Compression Method	uint8_t	8
FLG	gzip FLaGs	uint8_t	4
MTIME	gzip Modification TIME	uint32_t	
XFL	gzip eXtra FLags	uint8_t	
OS	gzip Operating System	uint8_t	
XLEN	gzip eXtra LENgth	uint16_t	
Extra subfield(s) (total size=XLEN)			
Additional RFC1952 extra subfields if present			
SI1	Subfield Identifier1	uint8_t	66
SI2	Subfield Identifier2	uint8_t	67
SLEN	Subfield LENgth	uint16_t	2
BSIZE	total Block SIZE minus 1	uint16_t	
Additional RFC1952 extra subfields if present			
CDATA	Compressed DATA by zlib::deflate()	uint8_t[BSIZE-XLEN-19]	
CRC32	CRC-32	uint32_t	
ISIZE	Input SIZE (length of uncompressed data)	uint32_t	

BAM is the binary representation of the SAM format. BAM is compressed in the BGZF format. The BGZF format is a series of blocks as shown in the table.

BGZF allows seeking efficiently within the data.

BSIZE defines the Block SIZE -1 and is only 16 bit long. Therefore BSIZE can represent BGZF block sizes in the range [1,65536].

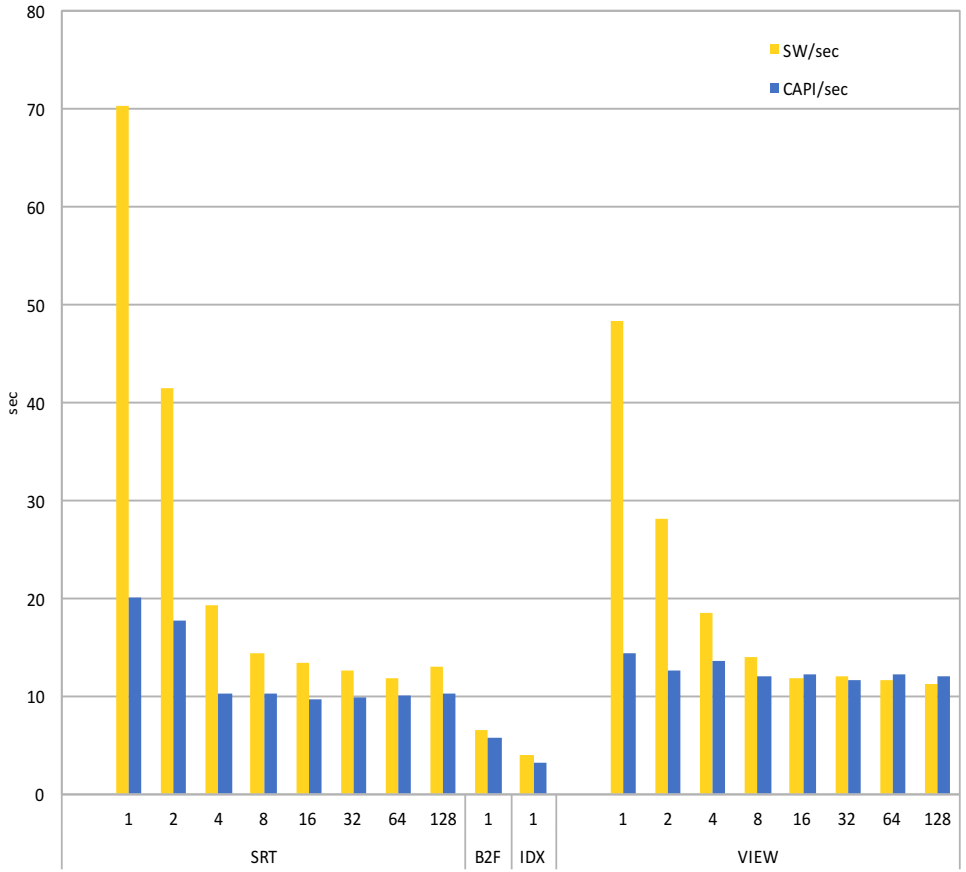
Blocksize of 64KiB limits the effectiveness of the CAPI GZIP accelerator.

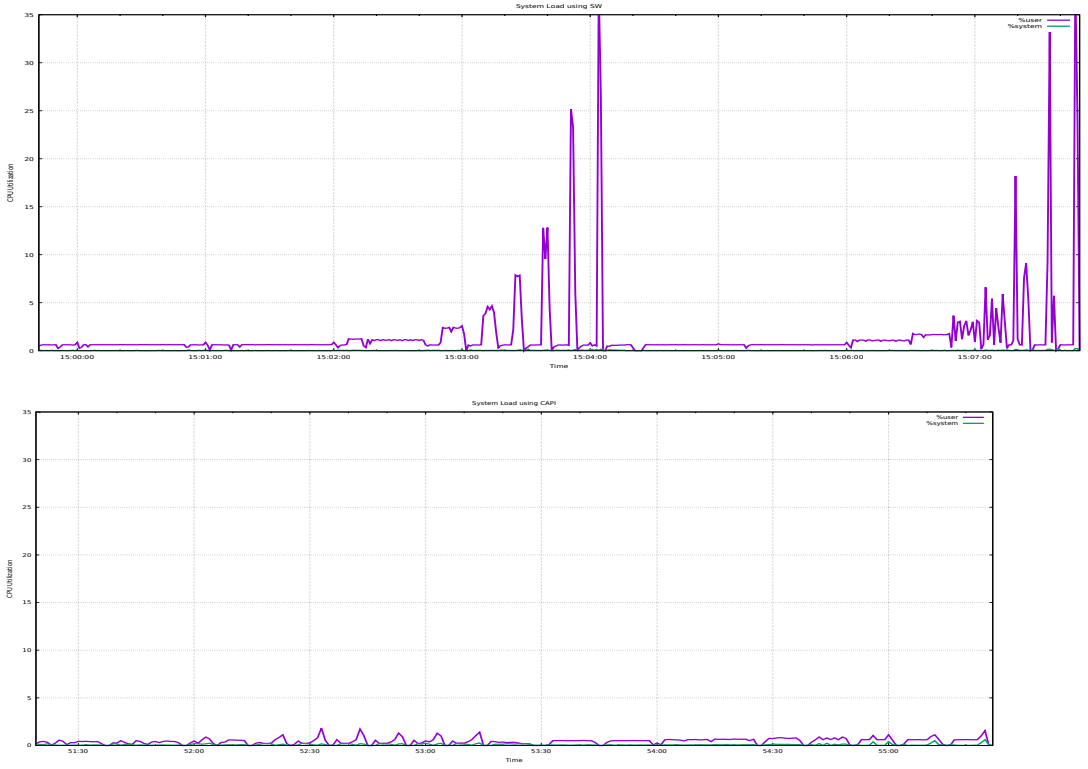
SAM: Sequence Alignment/Map Format
SAM/BAM file-format specification can be found here: https://samtools.github.io/hts-specs/SAMv1.pdf

Example 4 – Genomics: samtools

Samtools example

samtools with software zlib and CAPI GZIP acceleration





Top: CPU Usage with SW
Bottom: CPU Usage with CAPI GZIP

https://github.com/ibm-genwqe/genwqe-user/blob/master/misc/samtools_test.sh

Internal User Feedback

User feedback (Frank Liu, IBM Research):

„I measured the runtime on a larger file (5.4G), the performance difference is even more drastic: 11m41sec vs 4.7 sec. (...) But over 100x speedup is very impressive.“

The input data is SRR034947, half of about 1/40 of human genome sequence (the whole human genome sequence is split into about 40 parts, each part has two files, paired. This is one of the pair). The raw file size (uncompressed) is 5.4G. Compression reduced the file to 1.6G. The machine is a P8 Tuleta, S824, running RH 7.2.

How to use it?

Special Environment Variables

ZLIB_ACCELERATOR: Use CAPI or GENWQE (PCIe) cards. The default is GENWQE. In order to use a CAPI GZIP card with the accelerated zlib, set ZLIB_ACCELERATOR to CAPI.

NOTE: Most of our tools e.g. genwqe_gzip/genwqe_gunzip allow to switch between GenWQE and CAPI using the -A parameter instead of using the environment variable.

ZLIB_CARD: Card ID to be used. The library supports intelligent selection of cards when ZLIB_CARD is set to -1. Default setting is -1.

ZLIB_TRACE: Turn on tracing bits: 0x1: General, 0x2: Hardware, 0x4: Software, 0x8: Generate summary

ZLIB_DEFLATE_IMPL: Default setting is to use hardware.

0x00: SW, 0x01: HW The last 4 bits are used to switch between the hard- and software-implementation

0x40: Useful for cases like Genomics, or filesystems where the dictionary is not used after the operation completed, e.g. Z_FULL_FLUSH, etc. This is the default setting.

ZLIB_INFLATE_IMPL: Default setting is to use hardware, so you normally do not need to set it.

0x00: SW, 0x01: HW Use software or hardware for inflate

See above for control bits

ZLIB_LOGFILE: Printing out zlib traces on stderr is not appropriate e.g. if using zlib within a daemon, which closes stdin, stdout, and stderr use this to get the trace data.

The current list of possible environment variables is available here:

- https://github.com/ibm-genwqe/genwqe-user/wiki/Environment%20Variables

Simple steps to try out the PCIe/GenWQE card

1. Install required OS and genwqe-zlib, genwqe-tools packages
2. Figure out how zlib is used by your application (assume that it is linked dynamically for the example)
3. Execute your program without hardware accelerated zlib:

```
$ time scp -C linux.tar tul3:
linux.tar                                     100%  550MB  27.5MB/s   00:20
real      0m21.268s
user      0m20.780s
sys       0m0.432s
```

4. Execute your program with hardware accelerated zlib:

```
$ time ZLIB_ACCELERATOR=GENWQE LD_PRELOAD=/usr/lib/genwqe/libz.so.1 scp -C linux.tar tul3:
linux.tar                                     100%  550MB  91.7MB/s   00:06
real      0m6.248s
user      0m1.300s
sys       0m2.444s
```

```
$ time gzip -c linux.tar > linux.sw.tar.gz

real      0m17.248s
user      0m16.588s
sys       0m0.128s
$ time genwqe_gzip -AGENWQE -B0 -c linux.tar > linux.hw.tar.gz

real      0m1.217s
user      0m0.040s
sys       0m0.168s
```

Simple steps to try out the CAPI GZIP card

1. Install required OS and genwqe-zlib, genwqe-tools packages
2. Figure out how zlib is used by your application (assume that it is linked dynamically for the example)
3. Execute your program without hardware accelerated zlib:

$ time scp -C linux.tar tul3:

linux.tar 100% 550MB 26.2MB/s 00:21

real **0m21.276s**

user 0m20.720s

sys 0m0.508s

4. Execute your program with hardware accelerated zlib:

$ time ZLIB_ACCELERATOR=CAPI LD_PRELOAD=/usr/lib/genwqe/libz.so.1 scp -C linux.tar tul3:

linux.tar 100% 550MB 110.0MB/s 00:05

real **0m4.997s**

user 0m2.028s

sys 0m1.048s

$ time gzip -c linux.tar > linux.sw.tar.gz

real **0m16.730s**

user 0m16.580s

sys 0m0.124s

$ time genwqe_gzip -ACAPI -B0 -c linux.tar > linux.hw.tar.gz

real **0m1.221s**

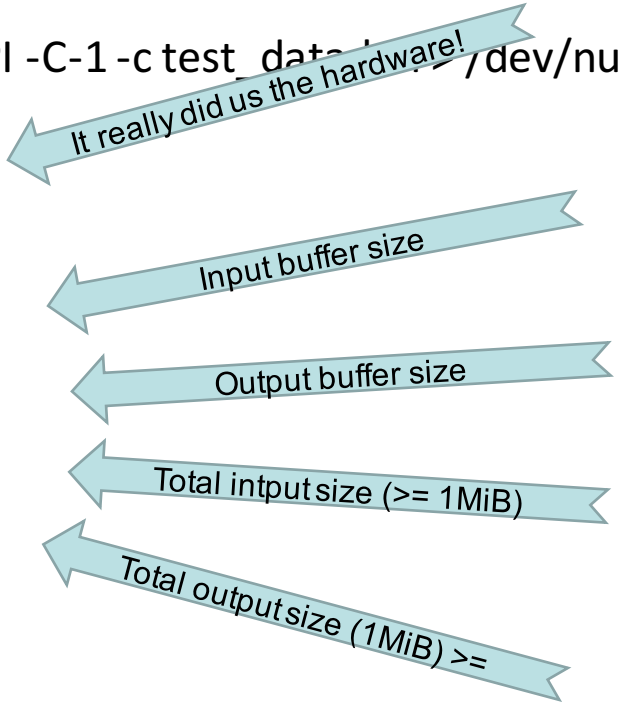
user 0m0.024s

sys 0m0.168s

Tracing zlib

The simplest way to use the trace is to let the library generate a summary of what it did:

```
$ ZLIB_TRACE=0x8 genwqe_gzip -ACAPI -C-1 -c test_data[illegible] > /dev/null
Info: deflateInit: 1
Info: deflate: 2242 sw: 0 hw: 2242
Info:   deflate_avail_in    4 KiB: 1120
Info:   deflate_avail_in   44 KiB: 1
Info:   deflate_avail_in  132 KiB: 1121
Info:   deflate_avail_out  132 KiB: 2242
Info:   deflate_total_in 1024 KiB: 1
Info:   deflate_total_out 1024 KiB: 1
Info: deflateSetHeader: 1
Info: deflateEnd: 1
Info: inflateInit: 0
Info: inflate: 0 sw: 0 hw: 0
Info: inflateEnd: 0
```

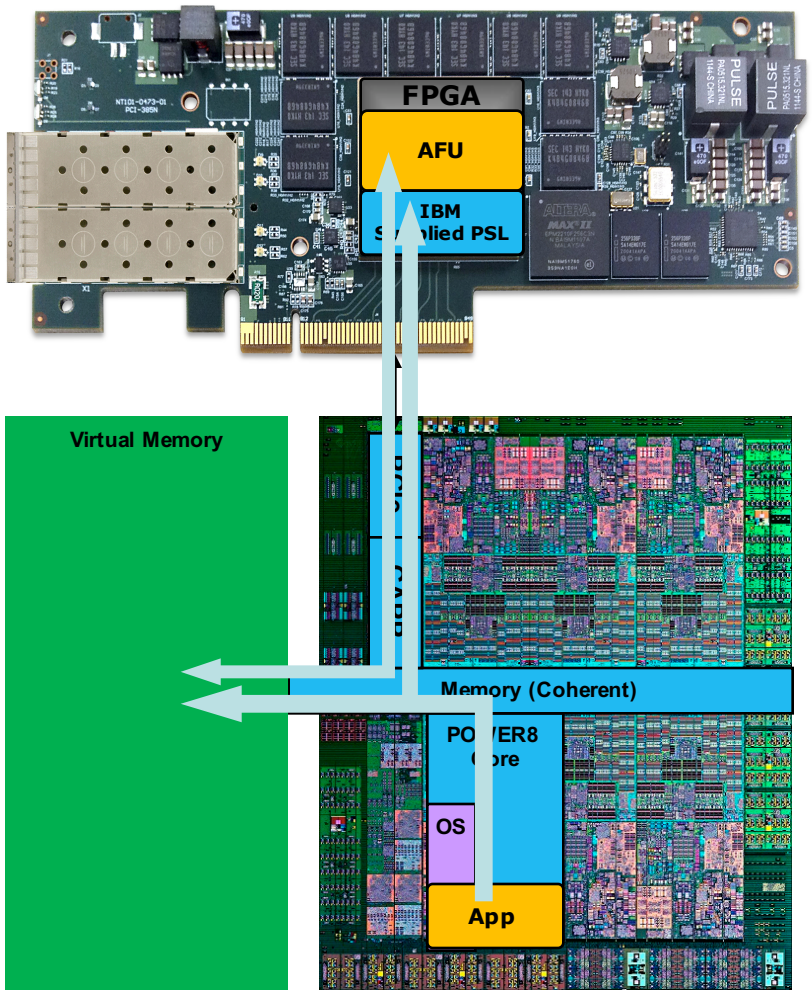


If the in and output buffer sizes are too small, consider enlarging it!

Coherent Accelerator Interface (CAPI)

- What is the Coherent Accelerator Interface (CAPI)?
- Accelerator Functional Unit (AFU) Modes
- Accessing AFU via Software

What is CAPI?



CAPI (Coherent Accelerator Processor Interface) is a set of IBM innovations in ***hardware*** and ***software*** that allow an ***application*** and it's ***accelerated component*** to share the same ***virtual address space***.

Proprietary hardware to enable coherent acceleration

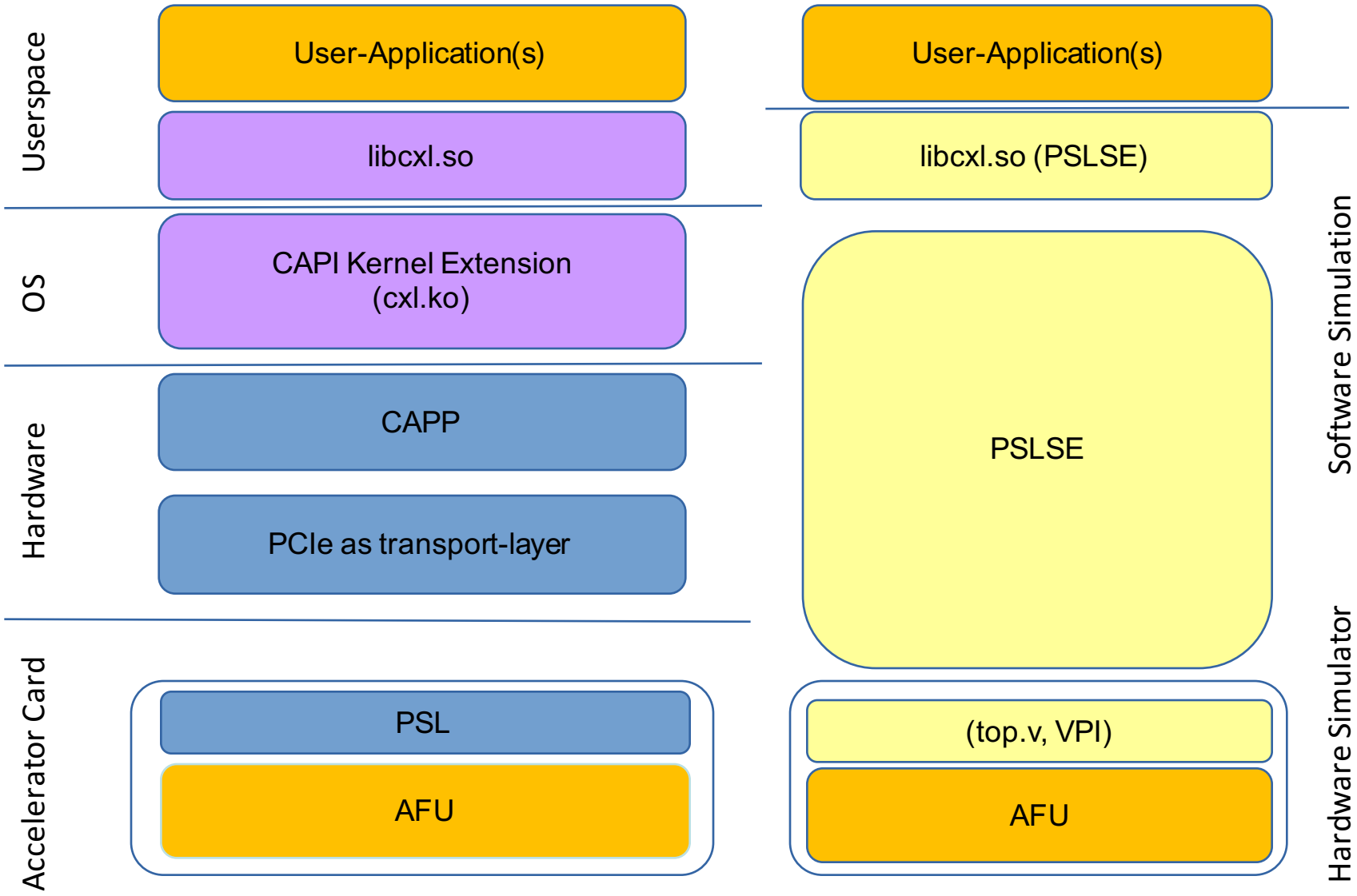
Operating system enablement
- Ubuntu LE/RHEL LE
- Libcxl function calls

Customer application and accelerator

For customers, CAPI enables performance and ease of programming:

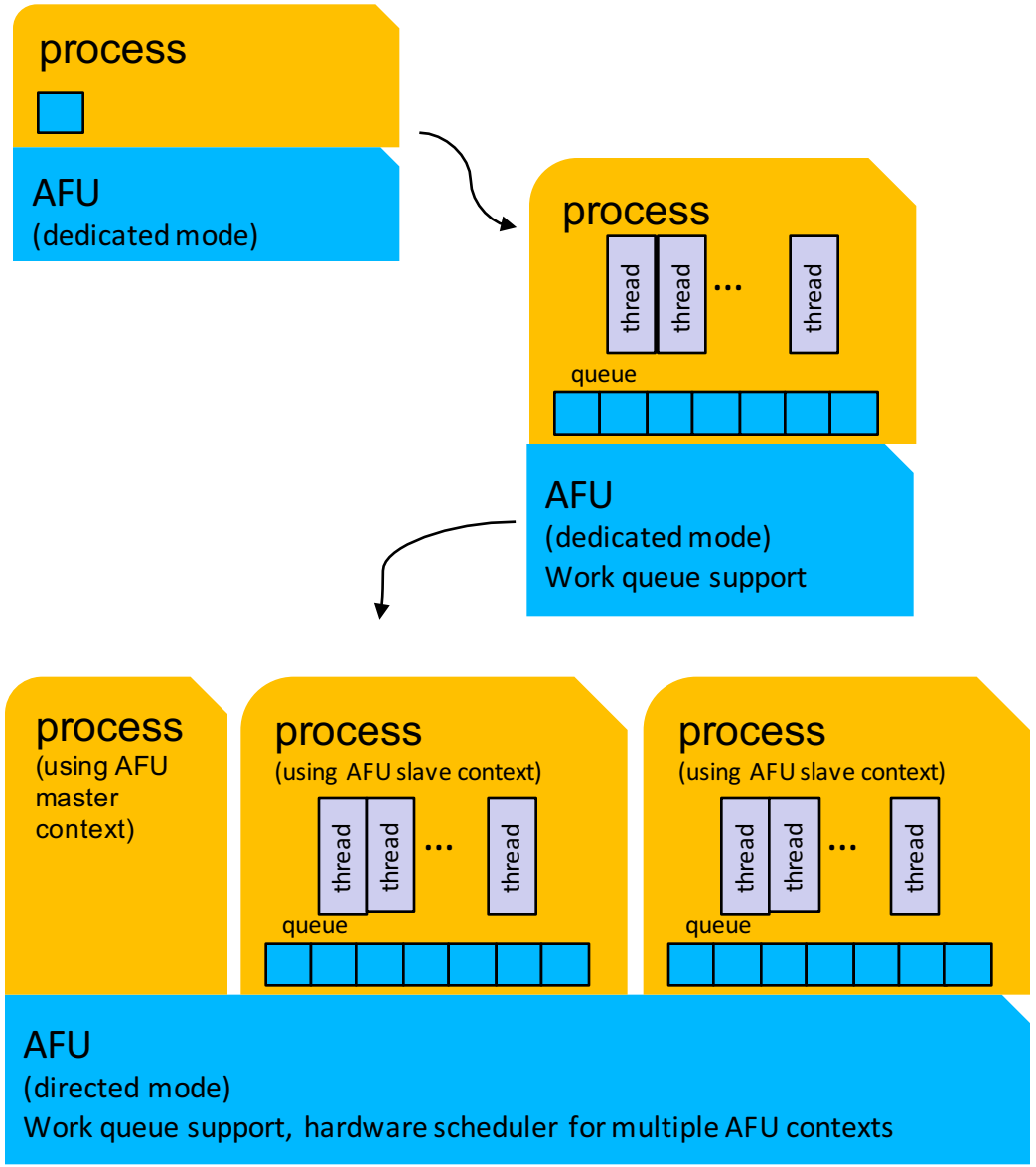
- Application to set up data coherently in shared virtual memory and call the accelerator functional unit (AFU)
- AFU to reads and writes data from and to shared virtual memory coherently across PCIe and communicating with the application
- AFU to coherently cache data in the PSL cache for quick AFU access

CAPI Layering



Accelerator Functional Unit (AFU) Modes

- Application: Single process
  - AFU dedicated mode: Single AFU context
  - Hardware provides: MMIO registers, Work Element Descriptor
- Application: Single process/multi-threaded
  - AFU dedicated mode: Single AFU context
  - Hardware provides: MMIO registers, Work-queue, interrupt
- Application: Multi process
  - AFU directed mode: Multiple AFU slave contexts, 1 AFU master context
  - Hardware provides: MMIO registers, Work-queues, interrupt(s) per AFU context



- **Summary:** CAPI GZIP supports up to 512 processes in parallel, which can have arbitrary number of threads

Accessing AFU with Software

- Open

```
afu_h = cxl_afu_open_dev("/dev/cxl/afu0.0s”);
rc = cxl_get_api_version_compatible(afu_h, &api_version_compatible);
rc = cxl_get_cr_vendor(afu_h, 0, &cr_vendor);
rc = cxl_get_cr_device(afu_h, 0, &cr_device);
afu_fd = cxl_afu_fd(afu_h);
rc = cxl_afu_attach(afu_h, (__u64)(unsigned long)(void *)w);
rc = cxl_mmio_map(afu_h, CXL_MMIO_BIG_ENDIAN);
```

- MMIO

```
rc = cxl_mmio_write64(afu_h, MMIO_DDCBQ_START_REG, &work_element_descriptor);
rc = cxl_mmio_write64(afu_h, MMIO_DDCBQ_COMMAND_REG, reg);
rc = cxl_mmio_read64(afu_h, MMIO_DDCBQ_STATUS_REG, &reg);
```

- Coherent memory access
  - Use “malloced” memory to communicate with accelerator

- Interrupts/events

```
rc = select(ctx->afu_fd + 1, &set, NULL, NULL, &timeout);
...
if (cxl_read_event(ctx->afu_h, &ctx->event) != 0)
            continue;
switch (ctx->event.header.type) {
case CXL_EVENT_AFU_INTERRUPT:
            while (__ddcb_done_post(ctx, DDCB_OK)); break;
case CXL_EVENT_DATA_STORAGE:
            /* do something */ break;
case CXL_EVENT_AFU_ERROR:
            /* do something */ break;
default:
            /* do something */ break;}
```

- Close

```
cxl_mmio_unmap(afu_h);
cxl_afu_free(afu_h);
```

Summary

- Use compression/decompression hardware accelerator
  - Providing high throughput
    (almost as fast as storing uncompressed data)
  - Saving storage space
  - Lowering CPU load
    (positive influence on power consumption)

- Available as PCIe and soon as CAPI version

- CAPI technology might be interesting for future own acceleration solutions

References/Links

GenWQE Hardware Accelerator Software

- Sources: https://github.com/ibm-genwqe/genwqe-user
- Wiki: https://github.com/ibm-genwqe/genwqe-user/wiki
- Bugtracking: https://github.com/ibm-genwqe/genwqe-user/issues

Packages

- Download and install genwqe-tools and genwqe-zlib packages
- RHEL7.x packages: https://www14.software.ibm.com/webapp/set2/sas/f/lopdiags/redhat/other/rhel7.html#
- Ubuntu (experimental yet): https://launchpad.net/~ibmpackages/+archive/ubuntu/genwqe/+packages

PCIe version of Generic Work Queue Engine (GenWQE)

- Application Programming Guide: https://www.ibm.com/developerworks/community/blogs/fe313521-2e95-46f2-817d-44a4f27eba32/entry/Generic_Work_Queue_Engine_GenWQE_Application_Programming_Guide?lang=en
- Generic Work Queue Engine (GenWQE) introduction: https://www.ibm.com/support/knowledgecenter/linuxonibm/liabt/liabtkickoff.htm

OpenPOWER™

- General Information: http://openpowerfoundation.org
- OpenPOWER™ Accelerator Work Group: http://openpowerfoundation.org/technical/working-groups/#wg_accelerator

Miscellaneous

- http://ethw.org/History_of_Lossless_Data_Compression_Algorithms
- http://zlib.net

BACKUP